ARMY RESEARCH LABORATORY

# API Design for a MicroTouch® Touch Screen Input Controller

Steven Choy

ARL-TR-2164 July 2000

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

**Army Research Laboratory**

Adelphi, MD 20783-1197

**ARL-TR-2164** **July 2000**

# API Design for a MicroTouch® Touch Screen Input Controller

Steven Choy

Computational and Information Sciences Directorate

## Abstract

TSLIB is a C application programming interface (API) for a flat-panel, touch screen input device manufactured by MicroTouch®. Unlike most software drivers for this device, the TSLIB interface is not designed to be a transparent replacement for a mouse input pointer. The TSLIB interface supports not only single-point input, but also continuous-point input to facilitate support for gesture-type input data in a multimodal, interactive environment. Also, unlike most software drivers for this device, the TSLIB software runtime support does not reside in the kernel/driver space of the operating system. The intent of the TSLIB design is to place the software interface driver in the user application space and connect to the physical device via the operating-system-supplied APIs for serial devices. Because the TSLIB API does not operate at the device driver level of the operating system, the interface source code can be easily ported across multiple hardware platforms and multiple software operating systems. The source code is designed for use on various UNIX platforms, including IRIX®, Solaris™, and LINUX, as well as under MS-DOS and Windows NT. A description of all API functions, data structures, and design algorithms is presented.

# Contents

# 1. Introduction

TSLIB is a C application programming interface (API) for a flat-panel, touch screen input device manufactured by MicroTouch®. Unlike most software drivers for this device, the TSLIB interface is not designed to be a transparent replacement for a mouse input pointer. Mouse input devices and their software interfaces are intended to be single-point-at-a-time input devices. The TSLIB interface supports not only single-point input, but also continuous-point input to facilitate support for gesture-type input data in a multimodal, interactive environment. Also, unlike most software drivers for this device, the TSLIB software runtime support does not reside in the kernel/driver space of the operating system. The intent of the TSLIB design is to place the software interface driver in the user application space and to connect to the physical device via the operating-system-supplied APIs for serial devices. Because the TSLIB API does not operate at the device driver level of the operating system, the interface source code can be easily ported across multiple hardware platforms and multiple software operating systems. The source code is designed to be used on various UNIX platforms, including IRIX®, Solaris™, and LINUX, as well as under MS-DOS and Windows NT. A detailed description of all API functions and data structures is presented. When the level of complexity of the logic warrants more detail, pertinent algorithms are shown within the particular functional description.

# 2. Technical Background of MicroTouch Touch Screen

Section 2.1 briefly describes the touch screen controller hardware interface. The message protocols for the controller are described in section 2.2, and a technical note regarding the touch screen coordinate system is presented in section 2.3.

## 2.1 The Touch Screen Hardware Interface

The touch screen controller interfaces to the CPU platform via a standard RS-232 serial data connection. The application library connects to the interface by calling the operating system interface functions that control communication with a serial device. The touch screen data packets are buffered at four levels:

1. The hardware device usually has a 2- to 32-byte data buffer, depending on the UART.
2. The operating system usually has a dynamically resizable data buffer.
3. TSLIB has a 512-byte input/output (I/O) data buffer.
4. TSLIB has a 5000-point data buffer, which is good for at least 25 s of continuous, nonstop pointing at 9600 baud.

On hardware reset, the controller takes on the following initial conditions:

- serial port: no parity, 7 data bits, 2 stop bits, and 9600 baud;
- autobaud detection enabled; and
- data packet protocol: mode stream and format decimal.

TSLIB software reset changes the initial conditions (as suggested by MicroTouch) to

- serial port: no parity, 8 data bits, 1 stop bit, and 9600 baud;
- autobaud detection disabled; and
- data packet protocol: mode stream and format tablet.

## 2.2 General Protocol

The MicroTouch controller uses two types of communication packet formats:

**Type 1**—The control packet is used to establish controller configuration parameters. This type of packet is used almost exclusively for setup and initialization of the controller. TSLIB will send a control packet of the form

```
<SOH>command<CR> ,
```

where the command is a short ASCII string indicating the type of operation to perform.

The controller sends a reply back to TSLIB in the same format, where the command is

- "0," indicating successful command completion;
- "1," indicating failure; or
- some other text string, indicating an informational reply.

For example, to request controller identity, TSLIB sends <SOH>OI<CR>, and the controller replies <SOH>A30550<CR> .

**Type 2**—The data packet is used to send touch screen coordinates from the controller to TSLIB. The packet is always 5 bytes long, represented as SXxYy and interpreted as follows:

- S0, S1—touch pen (not used in this software);
- S2–S4—reserved;
- S5—pen or finger (not used);
- S6—1 = PenDown, 0 = PenUp;
- S7—always 1 (sync bit);
- X0–X6 = lsb, x0–x6 = msb; and
- Y0–Y6 = lsb, y0–y6 = msb.

## 2.3 The Touch Screen Coordinate System

Under TSLIB, the touch screen controller returns data values that have 14-bit resolution and fall in the range of 0 to 16383. The origin (0,0) is located in the lower left corner of the display screen. Each data point collected by the controller can have one of three states:

1. PenDown—The finger is now touching the screen after previously being in the PenUp state (i.e., the first point of a line);
2. PenContinue—The finger is now touching the screen after previously being in the PenDown state (i.e., the *i*th point of a line, but not the first point); or
3. PenUp—The finger is not touching the screen after previously being in the PenContinue state (i.e., the last point of a line).

# 3. TSLIB—Touch Screen API Descriptions

The touch screen API is presented as both a user's reference document and a reference for the algorithmic design. Section 3.1 shows a simple example of API usage in a typical processing thread. Section 3.2 introduces two new data types established by this API, and section 3.3 lists all API methods, including calling syntax and semantics, as well as basic design algorithms.

## 3.1 Summary of a Typical Touch Screen Application Thread

A minimal touch screen application thread would use the TSLIB API in the following manner:

1. Call `tsOpen` to create an instance of a `TouchScreen` on a particular RS232 port. Call `tsClose` to delete the instance.
2. Call `tsSetScreenSize` to set the scaling into some user pixel coordinate system. Other tsSetXxxx functions can be optionally invoked for modifying the behavior of the TouchScreen device.
3. Call `tsOnPenUp` to set up user callback for a PenUp (called when finger lifts from the screen) event. Other callbacks are available for `PenDown` and `PenContinue` events.
4. Call `tsAppLoop` to wait for data. Typically, this routine will never return unless an error occurs or `tsSetAppLoopAbort` is invoked. The routine `tsSetAppMode` can be used to modify this behavior.
5. When a finger is lifted, the user callback routine will receive the `TouchScreen context`, a `userArgument` (uArg, typically the user application context), the $x$, $y$ coordinate in user units of the last point, and the number of points that were collected (after data filtering) between the PenDown and the PenUp events. If desired, `tsGetLine` can be called to retrieve the entire line and `tsGetPenTime` can be used to determine the time at which the line was drawn.

   For example, the sample program that follows attaches to the touch screen controller on UNIX serial port `/dev/ttya`. It sets the user screen units to $1280 \times 1024$ pixels. The data filter is set to require the pen (finger) to move at least five pixels in either the $x$ (horizontal) or $y$ (vertical) direction before recording the data point. The pixel location and the number of points drawn from PenDown to PenUp is printed every time the pen is lifted from the screen. If the user touches the lower right corner of the screen, the program exits.

```
#include <stdio.h>
#include <touchscreen.h>

/* pen UP callback handler */
void PenUpCB( TouchScreen h, int unused, TSPoint pt, int ln )
     {
     printf("PenUp X=%d, Y=%d, lineLen=%d )\n", pt.x, pt.y, ln
     );
     if( pt.x >1260 && pt.y < 20 )tsSetAppLoopAbort( h );
     }

main( int argc, char **argv )
     {
     TouchScreen h;
     h = tsOpen( "/dev/ttya" );
     if( h != NULL )
         {
        /* set scaling, filter, PenUp callback */
         tsSetScreenSize( h, 1280, 1024 );
         tsSetDataFilter( h, 5 );
         tsOnPenUp( h, PenUpCB, 0 );
        /* process touch screen events *
         tsAppLoop( h );
         tsClose( h );
         }
     }
```

Note: Additional examples can be found in the test code examples located in source files tstest1.c and tstest2.c (see sect. 5.2).

## 3.2 TSLIB Data Types

TSLIB introduces two new data types, TouchScreen and TSPoint. Each type is described below in terms of its individual components. When applicable, the component initialization is shown in parentheses immediately following the component name.

### *3.2.1 TouchScreen*

**Description:** Context for a touch screen object (file: touchscreen.h).

**Components**

*I/O context*

TTYCTX port (init: NULL)—I/O port context

char iobuf[512] (init: NONE)—input buffer for information from device

int iobufLen (init: 0)—number of entries in iobuf

char iostat[512] (init: iostat[0] = 0)—NULL-terminated string with the last response from the device

int lostSync (init: 0)—count of lost data packets

*Scaling and filtering behavior and line attributes*

int filter (init: 1)—line filter

int scrX, scrY (init: 16383, 16383, i.e., 3fff, 3fff)—maximum number of pixels on screen

float scaleX, scaleY (init: 1.0, 1.0 )—scale factor to get to user scaled units

(Note: No translation assumes that the origin of the user space is always (0,0).)

TSPoint line[5000] (init: NONE)—internal line buffer for current line

(Note: Allows for, at most, 25 s. If line gets full, then generate an auto PenUp.)

int lineLen (init: 0)—current length of line

int timePenDown (init: 0)—time when pen went down

int timePenUp (init: 0)—time when pen went up

int lineID (init: 0)—current line ID (probably just an index counter)

int penState (init: 1)—current pen state

1—TSPenStateUp, finger is up

2—TSPenStateDown, finger is down on first point

3—TSPenStateContinue, finger is down on *i*th point

*Callback*

int appMode (init: 1)—when to return from tsAppLoop

1—TSAppModeReturnOnlyOnError

2—TSAppModeReturnOnPenUp

3—TSAppModeReturnOnPenChange

int appAbortFlag (init: 0)—if not zero and PenUp then abort AppLoop

void (*userCBPenDown)(TouchScreen h, int uArg, TSPoint now)

(init: NULL)—user callback, PenDown

void (*userCBPenContinue)(TouchScreen h, int uArg, TSPoint now)

(init: NULL)—user callback, PenContinue

void (*userCBPenUp)(TouchScreen h, int uArg, TSPoint now, int lineLen)

(init: NULL)—user callback, PenUp

int PenDownCBArg, int PenContinueCBArg, PenUpCBArg—user CB Arg

### 3.2.2 *TSPoint*

**Description:** touch screen data point used in a line (file: tspoint.h).

**Components**

int x—*x* coordinates of a point in user scaled units

int y—*y* coordinates of a point in user scaled units

## 3.3 TSLIB User API Functions

Individual function descriptions of the API given here are sorted alphabetically by function name.

---

`int tsAppLoop(TouchScreen h)`

**Description:** Wait for data events from the touch screen.

**Returns:** This function normally runs forever. The routine tsSetAppMode can change this.

**Algorithm**

*Local variables*

`int status`—status returned from getting next point buffer

`TSPoint pt`—point just returned in user units

`TSPoint nowPt`—previous point returned in user units

```
in a loop process data points
   if penState is PenUp and abortFlag is set then
      return with appLoopAbort error
   get next point buffer
   if status indicates not a point return error (never
   happens!)
   pt = convert point buffer to point in user space
   if current lineLen > 0 then
      nowPt = line[ lineLen-1]
      filter data based on distance between pt to nowPt
      if no change, then ignore this point(goto end of loop)
   if point is with pen up then
      set penState to PenUp
      store point into line[lineLen]
      increment lineLen
      record timePenUp
      call user callback if installed
      return if appmode requires it
   else if previous state was PenUp
      set penState to PenDown
      store point into line[0]
      lineLen = 1
      record timePenDown
      call user callback if installed
      return if appmode requires it
   else
```

```
      set penState to PenContinue
      store point into line[lineLen]
      increment lineLen
      if lineBuffer full then
           change penState to PenUp
      call user callback if installed
      return if appmode requires it
end of loop
```

---

`int tsClose(TouchScreen h)`

**Description:** Destroys touch screen.

**Returns:** If error, returns –1; else returns 0.

**Algorithm**

```
If context is not NULL then
    if I/O port is opened, then close it
    de-allocate context
```

---

`int tsDoCalibration(TouchScreen h)`

**Description:** Performs interactive internal calibration (needs standout for prompts), requiring user to poke first at the lower left corner of the screen and then at the upper right corner of the screen.

(**Note:** This function is probably not used by most applications.)

**Returns:** If error, returns –1; else returns 0.

---

`int tsGetFd(TouchScreen h)`

**Description:** Get file descriptor associated with I/O port.

**Returns:** If error, returns –1; else returns file descriptor.

---

`char *tsGetLastControllerMessage(TouchScreen h)`

**Description:** Get last ASCII text message from the MicroTouch controller.

**Returns:** Returns a pointer to a NULL-terminated ASCII text string.

---

`int tsGetLine(TouchScreen h, TSPoint *line, int maxline)`

**Description:** Get current line. The parameter maxline indicates the number of elements the user's receiving line buffer can hold.

**Returns:** If error, returns –1; else returns 0.

---

`int tsGetLineID(TouchScreen h)`

**Description:** Get ID of current line.

**Returns:** If no line, returns –1; else returns line ID.

---

`int tsGetLineLength(TouchScreen h)`

**Description:** Get current line length (number of points in line buffer since most recent PenDown).

**Returns:** If error, returns –1; else returns 0.

---

`int tsGetPenState(TouchScreen h)`

**Description:** Get current pen state.

**Returns:** PenStates are described below (then lists pen state descriptions).

---

`int tsGetPenTime(TouchScreen h, int *timeDown, int *timeUp)`

**Description:** Get the times associated with the last PenDown and PenUp events. Time information is to help support multimodal gesture correlation.

**Returns:** If error, returns –1; else returns 0.

---

`TSPoint tsGetPoint(TouchScreen h)`

**Description:** Get current point (or NULL if none).

**Returns:** If no point, returns (–99999, –99999); else returns $x$, $y$ of current point.

**Algorithm**

Allocate context

```
if lineLen > 0 then
    return line[lineLen-1]
else
    return (-99999,99999)
```

---

`int tsGetLostSyncCount(TouchScreen h)`

**Description:** Get count of data packets lost by interface.

**Returns:** If error, returns –1; else returns lostSync count.

---

`int tsOnPenContinue(TouchScreen h, void (*SvcHndl)(TouchScreen h, int uArg, TSPoint now), int uArg)`

**Description:** Set user callback function for PenContinue event. PenContinue is when the finger is touching the screen, and it is not the initial point.

**Returns:** If error, returns –1; else returns 0.

```
int tsOnPenDown(TouchScreen h, void (*SvcHndl)
(TouchScreen h, int uArg, TSPoint now), int uArg)
```

**Description:** Set user callback function for PenDown event. PenDown is when the user first touches the screen.

**Returns:** If error, returns –1; else returns 0.

---

```
int tsOnPenUp(TouchScreen h, void (*SvcHndl)
(TouchScreen h, int uArg, TSPoint now, int
lineLength), int uArg)
```

**Description:** Set user callback function for PenUp event. PenUp is when the finger is lifted off the screen.

**Returns:** If error, returns –1; else returns 0.

---

```
TouchScreen tsOpen(char *nameOfSerialPort)
```

**Description:** Create a new touch screen on some serial port.

**Returns:** If error, returns –1; else returns 0.

**Algorithm**

```
Allocate context
return error if memory allocation problem
initialize context to default state (see TouchScreen
datatype)
attach I/O Port
if error then
   deallocate context and return error
else return context
```

---

```
int tsReset(TouchScreen h)
```

**Description:** Send reset hardware commands to touch screen, which include

- hardware reset;
- autobaud disable;
- 9600 baud, no parity, 8 data bits, and 1 stop bit; and
- data format: stream mode and format tablet.

**Returns:** If error, returns negative value; else returns 0.

**(Note:** This function is probably not used by most applications.)

---

```
void tsSetAppLoopAbort(TouchScreen h)
```

**Description:** Set appAbortFlag so that tsAppLoop will return pen state equals PenUp.

**Returns:** Nothing is returned.

```
int tsSetAppMode(TouchScreen h, int appMode)
```

**Description:** Set behavior for tsAppLoop. See description above for possibilities.

**Returns:** If error, returns –1; else returns 0.

(**Note:** This routine is probably never used. It is provided as an alternative to the callback mechanism.)

---

```
int tsSetDataFilter(TouchScreen h, int
distanceInScaledUnits)
```

**Description:** Do not store data in line unless $x$ or $y$ coordinate changes by distanceInScaledUnits.

(**Note:** A good value for a 1280 × 1024 screen seems to be between 5 and 10. Another possible filter would be the locking of the data to some user-specified grid increment and only passing through data that have changed a grid unit in one of the directions.)

**Returns:** If error, returns –1; else returns 0.

---

```
int tsSetScreenSize(TouchScreen h, int xmax, int ymax)
```

**Description:** Set size of screen in pixels for scaled units.

**Returns:** If error, returns –1; else returns 0.

**Algorithm**

```
Save new xmax and ymax into context
Calculate scaleX and scaleY by dividing
   each max by fullscale (i.e. 14 bits, 16383, 0x3fff)
```

## 3.4 TSLIB Low-Level Functions

---

```
int tsDistance( int p1, int p2, int distance)
```

**Description:** Determines if difference between p1 and p2 is greater than distance.

**Returns:** If absoluteValue($a - b$) is greater than or equal to dist, returns 1; else returns 0.

TSBufToPoint( char *bf, float scaleX, float scaleY )

**Description:** Converts Microtouch table formatted data to TSPoint.

**Note:** MicroTouch data format is SXxYy, where

- S0, S1 for touch pen (not used);
- S2–S4—reserved;
- S5—pen or finger (not used);

- S6—1 = touchDown, 0 = touchUp;
- S7 always 1 (sync bit);
- X0–X6 = lsb, x0–x6 = msb; and
- Y0–Y6 = lsb, y0–y6 = msb.

**Returns:** Converts *x* and *y* to user units and returns the point.

**Algorithm**

```
pt.x = (int)(.5 + (scaleX * (float)tsEvalBuffer( bf+1 )))
pt.y = (int)(.5 + (scaleY * (float)tsEvalBuffer( bf+3 )))
```

---

```
int tsEvalBuffer(char *v)
```

**Description:** Converts 2-character buffer data to a 14-bit integer.

**Returns:** Computed 14-bit integer.

**Algorithm**

```
(((int)v[0]) & 0x7f) | ( ((int)v[1]) << 7)
```

---

```
tsGetPointBuffer(port, char *buf, int bufsize)
```

**Description:** Read a data packet from the I/O port. This version does error checking and recovers from loss data if the processor cannot keep up with the I/O rates (192 packets/s at 9600 baud).

**Returns:** –1 indicates an error (never occurs, because not implemented), and 0 indicates 5-byte data packet SXxYy, where

- S0, S1 for touch pen (not used);
- S2–S4—reserved;
- S5—pen or finger (not used);
- S6—1 = penDown, 0 = PenUp;
- S7 always 1 (sync bit);
- X0–X6 = lsb, x0–x6 = msb; and
- Y0–Y6 = lsb, y0–y6 = msb.

**Algorithm**

***Five states***

SyncData—looking for sync byte

LsbX—looking for lsbX

MsbX—looking for msbX

LsbY—looking for lsbY

MsbY—looking for msbY

```
state = SyncData
In a loop
  data = get next byte
  if error reading data return -1
  if data sign bit is set then
     state = SyncData
  if state is SyncData
  if data sign bit is set
      buf[0] = data
      state = LsbX
  else if state is LsbX
      buf[1] = data
      state = MsbX
  else if state is MsbX
      buf[2] = data
      state = LsbY
  else if state is LsbY
      buf[3] = data
      state = MsbX
  else if state is LsbX
      buf[4] = data
      return 0 (indicates point was found)
end loop
```

# 4. Touch Screen Utilities

Two low-level diagnostic utilities are provided for initializing the touch screen and for sending low-level messages to the hardware controller.

## 4.1 tsInit—To Initialize MicroTouch Touch Screen Controller

**Usage**: `tsInit <ttport> example: tsInit/dev/ttya`

**Algorithm**

```
get UNIX port name from command line arg[1]
open port
if error print error message and exit
send init sequence commands including
```

- `R - reset`
- `AD - disable autobaud`
- `PN812 - no parity, 8 data bits, 1 stop bit, 9600`
- `FT - use format tablet data mode`
- `MS - stream data`

```
close port
exit
```

## 4.2 tsDiag—To Talk to Touch Screen Controller

**Usage**: `tsDiag <ttport> example: tsDiag/dev/ttyb`

**Algorithm**

```
get UNIX port name from command line arg[1]
open port
if error print error message and exit
in a loop
    prompt and wait for keyboard command
    if command is QUIT then break loop
    else if command is DATA then
        wait and read one line segment (getData function)
    else if command is CX then
        send <SOH>CX<CR>
        wait for 3 replies (ack, lowerleft, upperright)
    else
        send SOH <command> <CR> to controller
        wait for reply
        Print reply
end loop
close port
exit
```

# 5. Source Code

The current version of the source code at the time of this writing is listed below, followed by some test programs for TSLIB.

## 5.1 TSLIB Source

```
  file: tslib1.c
  Date: Jan 14, 2000
  Jan 19, 2000 add TSAppLoopAbort, pen time

  touch screen application interface for MicroTouch controller

*/
#include <tspoint.h>
#include <unixtty.h>
#include <sys/types.h>
#include <time.h>

#define SYNCFLAG 0x80
#define SOH 1

#define StateSyncData 1
#define StateLsbX 2
#define StateMsbX 3
#define StateLsbY 4
#define StateMsbY 5

#define TSBufMax 512
#define TSLineMax 5000

#define TS_LIB_SYS_DEF
typedef struct tsctx
    {
    /* -I/O context */
    TTYCTX tty; /* I/O port */
    char iobuf[TSBufMax]; /* input buffer for information from device */
    int iobufLen; /* number of entries in iobuf */
    char iostat[TSBufMax]; /* null terminated string with last response from
device */
    int lostSync; /* stats on how much data is loss */

    /* -Scaling and filtering behavior and line attributes */
    int filter; /* line filter */
    int scrX, scrY; /* user spec. max pixels of screen */
    float scaleX, scaleY; /* scale factor to get to user scaled units */
    TSPoint line[TSLineMax]; /* internal line buffer for current line */
    /* note: If line gets full generate auto PenUp. */
    int lineLen; /* current length of line */
    int lineID; /* current line ID (probably just an index counter) */
    int penState; /* current pen state */
    int timePenDown; /* when finger touched screen */
    int timePenUp; /* when finger lifted from screen */
```

```
    /* -Callback */
    int appMode; /* when to return from tsAppLoop */
    int appAbortFlag; /* if not zero, abort appModeLoop on PenUP */
    void (*userCBPenDown)( struct tsctx *h, int uArg, TSPoint now );
    void (*userCBPenContinue)( struct tsctx *h, int uArg, TSPoint now );
    void (*userCBPenUp)( struct tsctx *h, int uArg, TSPoint now, int lineLen );
    int penDownCBArg, penContinueCBArg, PenUpCBArg;
    } TSCTX;

#include <touchscreen.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

TouchScreen tsOpen( char *nameOfSerialPort )
    {
    TouchScreen h;
    h = (TouchScreen)calloc( 1, sizeof( TSCTX ) );
    if( h == NULL )return NULL;

    h->iostat[0] = 0;
    h->iobufLen = 0;
    h->lostSync = 0;

    h->penState = TSPenStateUp;
    h->lineID = -1;
    h->lineLen = 0;
    h->timePenDown = 0;
    h->timePenUp = 0;
    h->scaleX = 1.0; h->scaleY = 1.0;
    h->scrX = TSFullScale; h->scrY = TSFullScale;
    h->filter = 1;

    h->appMode = TSAppModeReturnOnlyOnError;
    h->appAbortFlag = 0;
    h->userCBPenDown = NULL;
    h->userCBPenContinue = NULL;
    h->userCBPenUp = NULL;
    h->penDownCBArg = h->penContinueCBArg = h->PenUpCBArg = 0;

    h->tty = openTTY( nameOfSerialPort );
    if( h->tty == NULL )
        {
        free( h );
        h = NULL;
        }
    return h;
    }

int tsClose( TouchScreen h )
    {
    if( h != NULL )
        {
        if( h->tty != NULL )closeTTY( h->tty );
        free( h );
        }
    }
```

```
int tsReset( TouchScreen h )
    {
    if( 0 != tsPutCmdGetReply( h, "R" ) )return -1;
    if( 0 != tsPutCmdGetReply( h, "AD" ) )return -2;
    if( 0 != tsPutCmdGetReply( h, "PN812" ) )return -3;
    if( 0 != tsPutCmdGetReply( h, "FT" ) )return -4;
    if( 0 != tsPutCmdGetReply( h, "MS" ) )return -5;
    return 0;
    }

int tsDoCalibration( TouchScreen h )
    {
    printf("Begin TouchScreen low level calibration\n");
    if( 0 == tsPutCmdGetReply( h, "CX" ) )
        {
        printf("Touch screen lower left corner\n");
        getSOHLine( h->tty, h->iostat, TSBufMax );
        if( h->iostat[0] == '1' )
            {
            printf("Touch screen upper right corner\n");
            getSOHLine( h->tty, h->iostat, TSBufMax );
            if( h->iostat[0] == '1' )
                {
                printf("Calibration success\n");
                return 0;
                }
            }
        }
    printf("Calibration failed\n");
    return -1;
    }

/* warning: NO LOCK means set commands can occur while PEN is down */
int tsSetScreenSize( TouchScreen h, int xmax, int ymax )
    {
    /* note: no Xlate => origin of user space == (0,0) */
    h->scrX = xmax;
    h->scrY = ymax;
    h->scaleX = ((float)xmax)/TSFloatFullScale;
    h->scaleY = ((float)ymax)/TSFloatFullScale;
    }

int tsSetDataFilter( TouchScreen h, int distanceInScaledUnits )
    {
    h->filter = distanceInScaledUnits;
    }

int tsSetAppMode( TouchScreen h, int appMode )
    {
    h->appMode = appMode;
    }

int tsGetLineLength( TouchScreen h )
    {
    return h->lineLen;
    }
```

```
int tsGetLostSyncCount( TouchScreen h )
    {
    return h->lostSync;
    }

int tsGetLine( TouchScreen h, TSPoint *line, int maxline )
    {
    int i, n;
    TSPoint *p;
    if( h->lineLen > 0 )
        {
        n = maxline < h->lineLen ? maxline : h->lineLen;
        for( i=0, p = h->line; i<n; i++ )
            {
            *line++ = *p++;
            }
        }
    return h->lineLen;
    }

TSPoint tsGetPoint( TouchScreen h )
    {
    TSPoint noPoint;
    if( h->lineLen > 0 )return h->line[ h->lineLen-1 ];
    noPoint.x = TSNoPointExists;
    noPoint.y = TSNoPointExists;
    return noPoint;
    }

int tsGetPenState( TouchScreen h )
    {
    return h->penState;
    }

int tsGetLineID( TouchScreen h )
    {
    return h->lineID;
    }

int tsGetFd( TouchScreen h )
    {
    /* not implemented yet */
    return -1;
    }

char *tsGetLastControllerMessage( TouchScreen h )
    {
    return h->iostat;
    }

int tsOnPenDown( TouchScreen h, void (*SvcHndl)( TouchScreen, int, TSPoint ), int
uArg )
    {
    h->userCBPenDown = SvcHndl;
    h->penDownCBArg = uArg;
    }
```

```
int tsOnPenContinue( TouchScreen h, void (*SvcHndl)( TouchScreen, int, TSPoint ),
int uArg )
    {
    h->userCBPenContinue = SvcHndl;
    h->penContinueCBArg = uArg;
    }

int tsOnPenUp( TouchScreen h, void (*SvcHndl)( TouchScreen, int, TSPoint, int ),
int uArg )
    {
    h->userCBPenUp = SvcHndl;
    h->PenUpCBArg = uArg;
    }

static int tsDistance( int a, int b, int dist )
    {
    int d = a-b;
    if( d<0)d = -d;
    return (d>=dist) ? 1:0;
    }

void tsSetAppLoopAbort( TouchScreen h )
    {
    h->appAbortFlag = 1;
    }

void tsGetPenTime( TouchScreen h, int *down, int *up )
    {
    *down = h->timePenDown;
    *up = h->timePenUp;
    }

int tsAppLoop( TouchScreen h )
    {
    int status;

    for(;;)
        {
        TSPoint pt, now;
        int pen;

        if( (h->penState == TSPenStateUp) &&
            (h->appAbortFlag != 0 ) )return TSAppLoopAbort;

        status = tsGetPointBuffer( h, h->iobuf, TSBufMax );
        if( status==0 )
           {
           pt = tsBufToPoint( h->iobuf, h->scaleX, h->scaleY );
           pen = h->iobuf[0];
           if( (h->lineLen > 0) && (pen != 0) )
               {
               now = h->line[h->lineLen-1];
               if( tsDistance( pt.x, now.x, h->filter ) == 0 &&
                   tsDistance( pt.y, now.y, h->filter ) == 0 )continue;
               }
           if( pen == 0 )
               {
```

```
            h->timePenUp = (int)time(0);
            h->penState = TSPenStateUp;
            h->line[ h->lineLen++ ] = pt;
            if( h->userCBPenUp != NULL )
                {
                (*h->userCBPenUp)( h, h->PenUpCBArg, pt, h->lineLen );
                }
            if( h->appMode == TSAppModeReturnOnPenUp )return 0;
            if( h->appMode == TSAppModeReturnOnPenChange )return 0;
            }
        else if( h->penState == TSPenStateUp )
            {
            h->timePenDown = (int)time(0);
            h->penState = TSPenStateDown;
            h->line[ 0 ] = pt;
            h->lineLen = 1;
            h->lineID += 1;
            if( h->userCBPenDown != NULL )
                {
                (*h->userCBPenDown)( h, h->penDownCBArg, pt );
                }
            if( h->appMode == TSAppModeReturnOnPenChange )return 0;
            }
        else
            {
            h->line[ h->lineLen++ ] = pt;
            /* over flow check on continue */
            if( h->lineLen >= TSLineMax )
                {
                h->penState = TSPenStateUp;
                if( h->userCBPenUp != NULL )
                    {
                    (*h->userCBPenUp)( h, h->PenUpCBArg, pt, h->lineLen );
                    }
                if( h->appMode == TSAppModeReturnOnPenUp )return 0;
                }
            else
                {
                h->penState = TSPenStateContinue;
                if( h->userCBPenContinue != NULL )
                    {
                    (*h->userCBPenContinue)( h, h->penContinueCBArg, pt );
                    }
                }
            if( h->appMode == TSAppModeReturnOnPenChange )return 0;
            }
        }
    else
        {
        /* Not a data point, probably never happen */
        strncpy( h->iostat, h->iobuf, status );
        /* maybe this is an error so return???? */
        return status;
    }
```

```
/* ---------- low level utilities ---------- */

int tsEvalBuffer( char *v )
    {
    return ( ((int)v[0]) & 0x7f) | ( ((int)v[1]) << 7);
    }

TSPoint tsBufToPoint( char *bf, float scaleX, float scaleY )
    {
    TSPoint pt;
    pt.x = (int)(.5 + (scaleX * (float)tsEvalBuffer( bf+1 )));
    pt.y = (int)(.5 + (scaleY * (float)tsEvalBuffer( bf+3 )));
    return pt;
    }

int tsGetPointBuffer( TouchScreen h, char *bf, int lenbuf )
    {
    int state;
    int val;
    int lossFlag = 1;
    int lossCount = 0;

    state = StateSyncData;
    for(;;)
        {
        val = wgetio( h->tty );
        if( (SYNCFLAG & val) != 0 )state = StateSyncData;
        switch( state )
            {
        case StateSyncData:
            if( (SYNCFLAG & val) != 0 )
                {
                bf[0] = (val >> 6) & 1;
                state = StateLsbX;
                lossFlag = 1;
                lossCount = 0;
                }
            else /* this is optional logic to keep track of lost data */
                {
                h->lostSync += lossFlag;
                if( lossFlag == 0 )
                    {
                    lossCount += 1;
                    if( lossCount > 4)
                        {
                        lossCount = 0;
                        h->lostSync += 1;
                        }
                    }
                lossFlag = 0;
                }
            break;
        case StateLsbX:
            bf[1] = val;
            state = StateMsbX;
            break;
        case StateMsbX:
```

```
            bf[2] = val;
            state = StateLsbY;
            break;
        case StateLsbY:
            bf[3] = val;
            state = StateMsbY;
            break;
        case StateMsbY:
            bf[4] = val;
            return 0;
            }
        }
    }
int tsPrintDataBuffer( char *bf )
    {
    int s = bf[0];
    int x = tsEvalBuffer( bf+1 );
    int y = tsEvalBuffer( bf+3 );
    if( s == 0 )printf("Up, ");
    else printf("Dn, ");
    printf("%x, %x - %x %x %x %x %x\n", x, y,
        0xff & (int)bf[0],
        0xff & (int)bf[1],
        0xff & (int)bf[2],
        0xff & (int)bf[3],
        0xff & (int)bf[4] );
    }

int tsPutCmdGetReply( TouchScreen h, char *cmd )
    {
    putSOHLine( h->tty, cmd );
    getSOHLine( h->tty, h->iostat, TSBufMax );
    return h->iostat[0] - '0';
    }

/* -------------------

   file: touchscreen.h
   Date: Jan 14, 2000
         Jan 19, 2000 add TSAppLoopAbort, penTime

   touch screen application interface for MicroTouch controller

*/
#ifndef TS_INCLUDE_DEF_
#define TS_INCLUDE_DEF_

#include <tspoint.h>

/* pen states */
#define TSPenStateUp 1
#define TSPenStateDown 2
#define TSPenStateContinue 3

/* app loop behavior */
#define TSAppModeReturnOnlyOnError 1
#define TSAppModeReturnOnPenUp     2
```

```
#define TSAppModeReturnOnPenChange 3

/* data values are 14 bits */
#define TSFullScale 0x3fff
#define TSFloatFullScale ((float)TSFullScale)
#define TSNoPointExists -99999
#define TSAppLoopAbort  -99999

#ifndef TS_LIB_SYS_DEF
typedef struct tsctx{int dummy;} TSCTX;
#endif

typedef TSCTX * TouchScreen;

#ifdef __cplusplus
extern "C" {
#endif

TouchScreen tsOpen( char *nameOfSerialPort );
int tsClose( TouchScreen h );
int tsReset( TouchScreen h );
int tsDoCalibration( TouchScreen h );
int tsSetScreenSize( TouchScreen h, int xmax, int ymax );
int tsSetDataFilter( TouchScreen h, int distanceInScaledUnits );
int tsSetAppMode( TouchScreen h, int appMode );
int tsGetLineLength( TouchScreen h );
int tsGetLine( TouchScreen h, TSPoint *line, int maxline );
TSPoint tsGetPoint( TouchScreen h );
int tsGetPenState( TouchScreen h );
void tsGetPenTime( TouchScreen h, int *down, int *up );
int tsGetLineID( TouchScreen h );
int tsGetFd( TouchScreen h );
char *tsGetLastControllerMessage( TouchScreen h );
int tsGetLostSyncCount( TouchScreen h );
int tsOnPenDown( TouchScreen h, void (*SvcHndl)( TouchScreen h,
                 int uArg, TSPoint now ), int uArg );
int tsOnPenContinue( TouchScreen h, void (*SvcHndl)( TouchScreen h,
                 int uArg, TSPoint now ), int uArg );
int tsOnPenUp( TouchScreen h, void (*SvcHndl)( TouchScreen h,
                 int uArg, TSPoint now, int lineLength ), int uArg );
int tsAppLoop( TouchScreen h );
void tsSetAppLoopAbort( TouchScreen h );

/* low level utilities */
int tsEvalBuffer( char *v );
int tsGetPointBuffer( TouchScreen h, char *bf, int lenbuf );
int tsPrintDataBuffer( char *bf );
int tsPutCmdGetReply( TouchScreen h, char *cmd );
TSPoint tsBufToPoint( char *bf, float scaleX, float scaleY );
static int tsDistance( int a, int b, int dist );

#ifdef __cplusplus
}
#endif

#endif
```

## 5.2 Some Test Programs for TSLIB

```
   file: tstest1.c
   date: Jan 14, 2000
   Jan 19, 2000 add TSAppLoopAbort

  quick api test

  prints x,y on every pen up

*/
#include <stdio.h>
#include <stdlib.h>
#include "touchscreen.h"

void PenUpCB( TouchScreen h, int unused, TSPoint pt, int ln );

main( int argc, char **argv )
    {
    TouchScreen h;
    h = tsOpen( "/dev/ttya" );
    printf( "tstest1, pen up test\n");
    if( h != NULL )
        {
        tsSetScreenSize( h, 1280, 1024 );
        tsSetDataFilter( h, 5 );
        tsOnPenUp( h, PenUpCB, 0 );
        tsAppLoop( h );
        tsClose( h );
        }
    printf("Bye, bye\n");
    exit( 0 );
    }

void PenUpCB( TouchScreen h, int unused, TSPoint pt, int ln )
    {
    int lost = tsGetLostSyncCount( h );
    printf("PenUp %d %d n=%d, loss=%d)\n", pt.x, pt.y, ln, lost );
    /* exit if we touch lower right corner */
    if( pt.x >1260 && pt.y < 20 )
        {
        tsSetAppLoopAbort( h );
        }
    }

/* --------------------

   file: tstest2.c
   date: Jan 14, 2000
   Jan 19, 2000 add TSAppLoopAbort

  2nd quick api test

  prints x,y on every pen change

  usage: tstest2 [filtervalue]
```

```
  default filtervalue is 5

*/
#include <stdio.h>
#include <stdlib.h>
#include "touchscreen.h"

/* test all 3 callback functions */
void PenUpCB( TouchScreen h, int ak, TSPoint pt, int ln );
void penDownCB( TouchScreen h, int ak, TSPoint pt );
void penContinueCB( TouchScreen h, int ak, TSPoint pt );

main( int argc, char **argv )
    {
    TouchScreen h;      /* touch screen context */
    int k;              /* fake user context for testing */
    int filter;         /* data filter distance */

    filter = 5;
    if( argc == 2 )sscanf( argv[1], "%d", &filter );
    h = tsOpen( "/dev/ttya" );
    printf( "tstest2, pen change test, filter =%d\n", filter );
    k = 0;
    if( h != NULL )
        {
        tsSetScreenSize( h, 1280, 1024 );
        tsSetDataFilter( h, filter );
        tsOnPenContinue( h, penContinueCB, (int)&k );
        tsOnPenDown( h, penDownCB, (int)&k );
        tsOnPenUp( h, PenUpCB, (int)&k );
        tsAppLoop( h );
        tsClose( h );
        }
    printf("Bye, bye\n");
    exit( 0 );
    }

/* ------------- call backs ******************/
void PenUpCB( TouchScreen h, int userArg, TSPoint pt, int ln )
    {
    int *k = (int *)userArg;
    *k += 1;
    printf("PenUp %d %d n=%d, lost=%d)\n", pt.x, pt.y, ln,
        tsGetLostSyncCount(h) );
    /* exit if we touch lower right corner */
    if( pt.x >1260 && pt.y < 20 )
        {
        tsSetAppLoopAbort( h );
        }
    }

void penDownCB( TouchScreen h, int userArg, TSPoint pt )
    {
    int *k = (int *)userArg;
    *k = 1;
    printf("PenDn %d %d)\n", pt.x, pt.y );
    }
```

```
void penContinueCB( TouchScreen h, int userArg, TSPoint pt )
    {
    int *k = (int *)userArg;
    *k += 1;
    printf("PenCt %d %d k=%d)\n", pt.x, pt.y, *k );
    }
```

# Distribution

Admnstr
Defns Techl Info Ctr
Attn DTIC-OCP
8725 John J Kingman Rd Ste 0944
FT Belvoir VA 22060-6218

Ofc of the Secy of Defns
Attn ODDRE (R&AT)
The Pentagon
Washington DC 20301-3080

Ofc of the Secy of Defns
Attn OUSD(A&T)/ODDR&E(R) R J Trew
3080 Defense Pentagon
Washington DC 20301-7100

AMCOM MRDEC
Attn AMSMI-RD W C McCorkle
Redstone Arsenal AL 35898-5240

Dir for MANPRINT
Ofc of the Deputy Chief of Staff for Prsnnl
Attn J Hiller
The Pentagon Rm 2C733
Washington DC 20301-0300

SMC/CZA
2435 Vela Way Ste 1613
El Segundo CA 90245-5500

TECOM
Attn AMSTE-CL
Aberdeen Proving Ground MD 21005-5057

US Army ARDEC
Attn AMSTA-AR-TD M Fisette
Bldg 1
Picatinny Arsenal NJ 07806-5000

US Army Info Sys Engrg Cmnd
Attn ASQB-OTD F Jenia
FT Huachuca AZ 85613-5300

US Army Natick RDEC Acting Techl Dir
Attn SSCNC-T P Brandler
Natick MA 01760-5002

US Army Simulation, Train, & Instrmntn Cmnd
Attn J Stahl
12350 Research Parkway
Orlando FL 32826-3726

US Army Soldier & Biol Chem Cmnd Dir of Rsrch & Techlgy Dirctrt
Attn SMCCR-RS I G Resnick
Aberdeen Proving Ground MD 21010-5423

US Army Tank-Automtv Cmnd Rsrch, Dev, & Engrg Ctr
Attn AMSTA-TR J Chapin
Warren MI 48397-5000

US Army Train & Doctrine Cmnd
Battle Lab Integration & Techl Dirctrt
Attn ATCD-B J A Klevecz
FT Monroe VA 23651-5850

US Military Academy
Mathematical Sci Ctr of Excellence
Attn MDN-A LTC M D Phillips
Dept of Mathematical Sci Thayer Hall
West Point NY 10996-1786

Nav Surface Warfare Ctr
Attn Code B07 J Pennella
17320 Dahlgren Rd Bldg 1470 Rm 1101
Dahlgren VA 22448-5100

DARPA
Attn S Welby
3701 N Fairfax Dr
Arlington VA 22203-1714

Hicks & Associates Inc
Attn G Singley III
1710 Goodrich Dr Ste 1300
McLean VA 22102

Director
US Army Rsrch Ofc
Attn AMSRL-RO-D JCI Chang
Attn AMSRL-RO-EN W D Bach
PO Box 12211
Research Triangle Park NC 27709

## Distribution (cont'd)

ElanTech, Inc.
4105 Nesconset Drive
Bowie MD 20716

MicroTouch Systems, Inc.
300 Griffin Brook Park Dr
Methuen MA 01844

US Army Rsrch Lab
Attn AMSRL-DD J M Miller
Attn AMSRL-CI-A R Rosen
Attn AMSRL-CI-A S Choy
Attn AMSRL-CI-AI-R Mail & Records Mgmt
Attn AMSRL-CI-AP Techl Pub (3 copies)
Attn AMSRL-CI-LL Techl Lib (3 copies)
Attn AMSRL-IS S Ho

US Army Rsrch Lab (cont'd)
Attn AMSRL-IS-CB B Klipple
Attn AMSRL-IS-CB D McCarthy
Attn AMSRL-IS-CB J Grills
Attn AMSRL-IS-CB J Gurney
Attn AMSRL-IS-CB L Tokarcik
Attn AMSRL-IS-CB R Winkler
Attn AMSRL-IS-CB T Gregory
Attn AMSRL-IS-CI C Winslow
Attn AMSRL-SE-RU M Conn
Attn AMSRL-SE-SA H Vu
Attn AMSRL-SE-SA M Fong
Attn AMSRL-WM-MB M Berman
Attn AMSRL-WM-MB T Li
Adelphi MD 20783-1197

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | July 2000 | Final, January–March 2000 |

**4. TITLE AND SUBTITLE** API Design for a MicroTouch® Touch Screen Input Controller

**5. FUNDING NUMBERS**
DA PR: N/A
PE: 61

**6. AUTHOR(S)** Steven Choy

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
U.S. Army Research Laboratory
Attn: AMSRL-CI-A email: schoy@arl.mil
2800 Powder Mill Road
Adelphi, MD 20783-1197

**8. PERFORMING ORGANIZATION REPORT NUMBER**
ARL-TR-2164

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
U.S. Army Research Laboratory
2800 Powder Mill Road
Adelphi, MD 20783-1197

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**
ARL PR: OUFT01
AMS code: 611102.H48

**12a. DISTRIBUTION/AVAILABILITY STATEMENT** Approved for public release; distribution unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

TSLIB is a C application programming interface (API) for a flat-panel, touch screen input device manufactured by MicroTouch®. Unlike most software drivers for this device, the TSLIB interface is not designed to be a transparent replacement for a mouse input pointer. The TSLIB interface supports not only single-point input, but also continuous-point input to facilitate support for gesture-type input data in a multimodal, interactive environment. Also, unlike most software drivers for this device, the TSLIB software runtime support does not reside in the kernel/driver space of the operating system. The intent of the TSLIB design is to place the software interface driver in the user application space and connect to the physical device via the operating-system-supplied APIs for serial devices. Because the TSLIB API does not operate at the device driver level of the operating system, the interface source code can be easily ported across multiple hardware platforms and multiple software operating systems. The source code is designed for use on various UNIX platforms, including IRIX®, Solaris™, and LINUX, as well as under MS-DOS and Windows NT. A description of all API functions, data structures, and design algorithms is presented.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| Touch screen, driver, API, UNIX, Windows | 35 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102